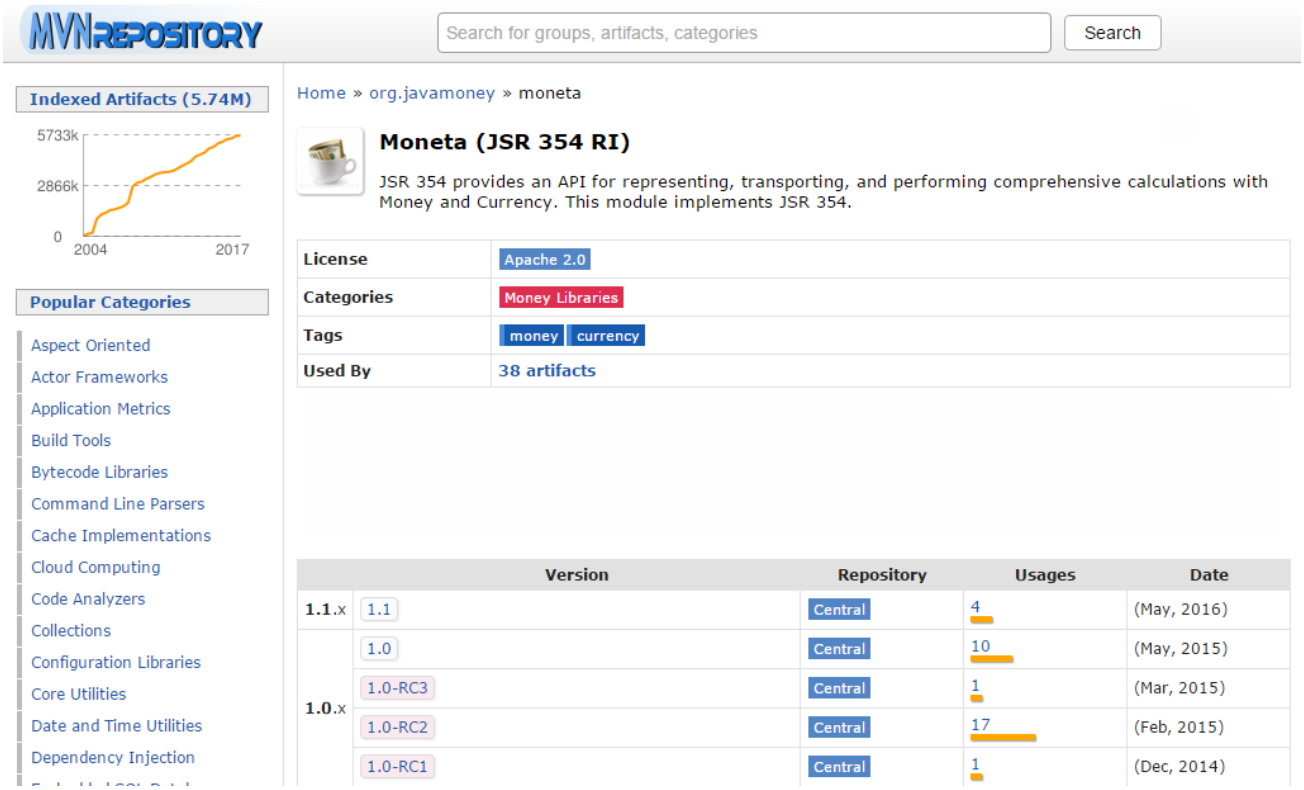
01

Javamoney

Transcrição

Frequentemente precisamos lidar com dinheiro em nossas aplicações, e como estamos no Brasil, especificamente o real. Já sabemos que, ao criar uma variável para dinheiro, é melhor trabalhar com `BigDecimal` do que com `double` por sua precisão.

Há uma API específica para trabalhar com dinheiro, a JSR 354 (https://jcp.org/en/jsr/detail?id=354), *Money and Currency API* . Ela ainda está sendo trabalhada e deve entrar na versão 9 do Java. Mas já conseguimos usá-la no Java nas aplicações do nosso cotidiano. Ela está disponível no GitHub (https://github.com/JavaMoney/jsr354-api), então você pode contribuir com o código. No curso, vamos usá-la pelo Maven (https://mvnrepository.com/artifact/org.javamoney/moneta). Nele, o nome dado para a implementação é Moneta, e usaremos a versão mais recente.



Copiaremos o código da dependência e a copiaremos para o `pom.xml` do nosso projeto, assim como fizemos com o código do Stella. O arquivo está assim:

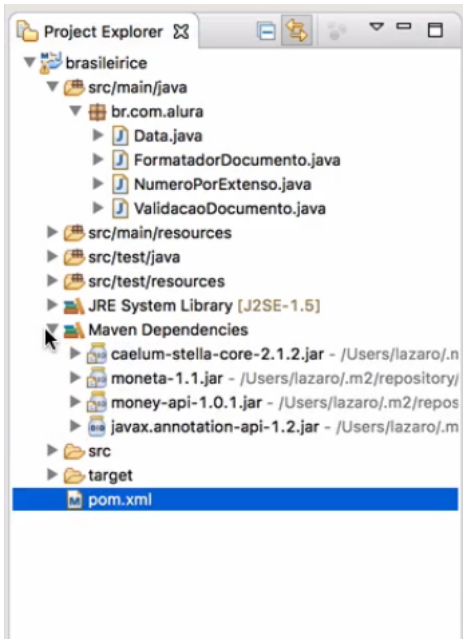
```
<project xmlns="http://maven.apache.org/POM/4.0.0""xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  <modelVersion>4.0.0</modelversion>
  <groupId>br.com.alura.brasileirice</groupId>
  <artifactId>Brasileirice</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <dependencies>
    <dependency>
        <groupId>br.com.caelum.stella</groupId>
        <artifactId>caelum-stella-core</artifactId>
```

```
        <version>2.1.2</version>
      </dependency>
  </dependencies>

</project>
```

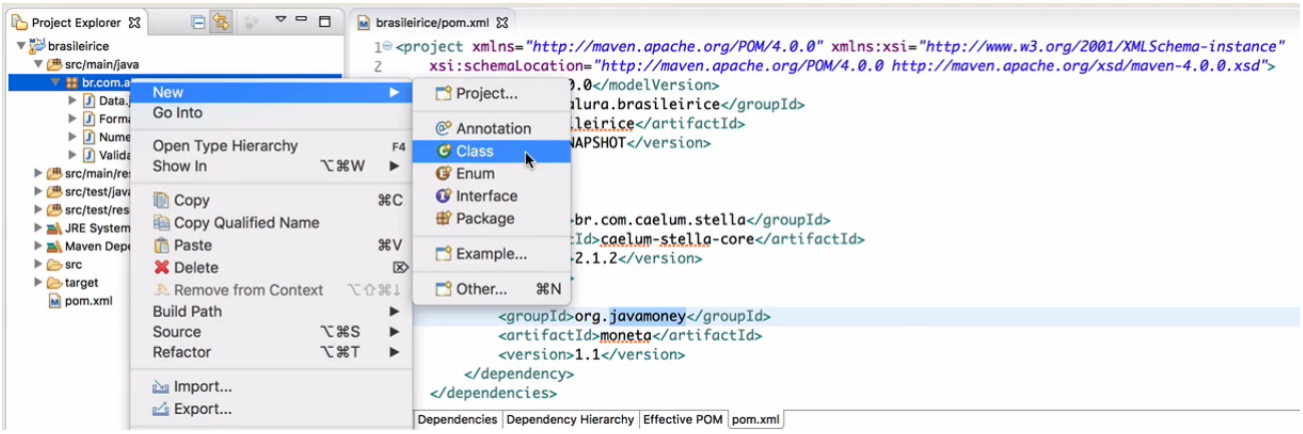
Acrescentando o código do Moneta:

```
<project xmlns="http://maven.apache.org/POM/4.0.0""xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  <modelVersion>4.0.0</modelversion>
  <groupId>br.com.alura.brasileirice</groupId>
  <artifactId>Brasileirice</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <dependencies>
    <dependency>
        <groupId>br.com.caelum.stella</groupId>
        <artifactId>caelum-stella-core</artifactId>
        <version>2.1.2</version>
      </dependency>
    <dependency>
      <groupId>org.javamoney</groupId>
      <artifactId>moneta</artifactId>
      <version>1.1</version>
    </dependency>
  </dependencies>

</project>
```

Ao salvar, o programa acrescentará os arquivos `moneta-1.1.jar` , `money-api-1.0.1.jar` e `javax.anootation-apri-1/2/jar` à nossa biblioteca.



Agora iremos criar uma nova classe, clicando com o botão direito sobre o projeto e em `New > Class` . Seu nome será `Dinheiro` .



Como de costume, começaremos criando o método `main`.

```
package br.com.alura;

public class Dinheiro {

  public static void main(String[] args){

  }

}
```

A primeira coisa a se fazer nessa API é definir a moeda com a qual trabalharemos. Usaremos o `Monetary` com o método `getCurrency()`. O real é o `"BRL"`.

```
package br.com.alura;

public class Dinheiro {

  public static void main(String[] args){
    Monetary.getCurrency("BRL");

  }

}
```

Criamos o que corresponde à moeda que usaremos, mas ela precisa ser uma instância do `CurrencyUnit`.

```
package br.com.alura;

public class Dinheiro {

  public static void main(String[] args){
    CurrencyUnit moeda = Monetary.getCurrency("BRL");

  }

}
```

Para criar um valor, usamos o `MonetaryAmount`. Usaremos o valor da parcela da Alura, dentro de `Money.of(number, currency)`, e já pediremos para imprimir no console.

```
package br.com.alura;

public class Dinheiro {

  public static void main(String[] args){
    CurrencyUnit moeda = Monetary.getCurrency("BRL");
    MonetaryAmount valorDaParcela = Money.of(75, moeda);
    System.out.prinln(valorDaParcela);
  }

}
```

Ao rodar, o console nos mostra o seguinte:

```
nov 07, 2016 3:32:33 PM org.javamoney.moneta.DefaultMonetaryContextFactory createMonetaryContext
INFORMAÇÕES: Using custom MathContext: precision=256, roundingMode=HALF_EVEN
BRL 75
```

As primeiras linhas são um aviso da API. Embaixo, temos o `BRL 75` que é a nossa moeda com o valor que determinamos. Troquemos por `USD`, que é o dólar, no código.

```
package br.com.alura;

public class Dinheiro {

  public static void main(String[] args){
    CurrencyUnit moeda = Monetary.getCurrency("USD");
    MonetaryAmount valorDaParcela = Money.of(75, moeda);
    System.out.prinln(valorDaParcela);
  }

}
```

Teremos no console:

```
nov 07, 2016 3:32:33 PM org.javamoney.moneta.DefaultMonetaryContextFactory createMonetaryContext
INFORMAÇÕES: Using custom MathContext: precision=256, roundingMode=HALF_EVEN
USD 75
```

Como estamos trabalhando com sistemas do Brasil, manteremos o `BRL`. Caso seja necessário multiplicar dólar por real, pode-se criar duas instâncias e fazer a multiplicação sem medo.

Estamos usando o `valorDaParcela`, e seria bom fazer uma multiplicação por 12 (pois é uma parcela mensal), para descobrir qual será o custo total. Precisamos criar um valor, que receberá o valor da parcela, multiplicado pelos 12 meses. A função que faz isso é a `multiply()`. Já pediremos um print no console.

```
package br.com.alura;

public class Dinheiro {

  public static void main(String[] args){
    CurrencyUnit moeda = Monetary.getCurrency("BRL");
    MonetaryAmount valorDaParcela = Money.of(75, moeda);
    System.out.prinln(valorDaParcela);
    MonetaryAmount valorTotal = valorDaParcela.multiply(12);
    System.out.println(valorTotal);
  }

}
```

Ao pedir para rodar, o console nos mostra:

```
nov 07, 2016 3:32:33 PM org.javamoney.moneta.DefaultMonetaryContextFactory createMonetaryContext
INFORMAÇÕES: Using custom MathContext: precision=256, roundingMode=HALF_EVEN
BRL 75
BRL 900
```

A conta foi feita para nós. Assim, se você precisar dividir, multiplicar ou somar, pode usar a classe `MonetaryAmount`. Essa é a nova maneira de trabalhar com dinheiro do Java, e esperamos que saia na versão 9. Mas já podemos ir usando quando necessário. Até a próxima!